

Drawing One-Hop Links Using the Common Open Research Emulator (CORE) Service

by Rommie L Hardy

ARL-TR-7097

September 2014

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory

Adelphi, MD 20783-1138

ARL-TR-7097

September 2014

Drawing One-Hop Links Using the Common Open Research Emulator (CORE) Service

Rommie L Hardy

Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
<p>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>					
1. REPORT DATE (DD-MM-YYYY) September 2014		2. REPORT TYPE Final		3. DATES COVERED (From - To) 07/2013–09/2013	
4. TITLE AND SUBTITLE Drawing One-Hop Links Using the Common Open Research Emulator (CORE) Service				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Rommie L Hardy				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-CIN-T 2800 Powder Mill Road Adelphi, MD 20783-1138				8. PERFORMING ORGANIZATION REPORT NUMBER ARL-TR-7097	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT <p>The current version of Common Open Research Emulator (CORE) used in the Network Science Research Laboratory only considers the distance-based formula to draw links on the graphical user interface (GUI) to represent network connectivity. This model works by calculating the pixel distance between two nodes positioned on the GUI surface then comparing the transposed pixel distance into meters whereby it can be compared to the maximum allowable distance between nodes. If the calculated distance is less than or equal to the maximum distance value, the link is drawn. This report describes a process for adding a service to CORE that will gather one-hop neighbor information based upon Open Shortest Path First (OSPF) and Optimal Link State Routing (OLSR) protocols. The network connectivity can then be displayed on the CORE GUI as well as other visualization tools used within the Network Science Research Laboratory based upon methods other than the distance-based formula.</p>					
15. SUBJECT TERMS CORE, Python, visualization					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 28	19a. NAME OF RESPONSIBLE PERSON Rommie L Hardy
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (Include area code) 301-394-1189

Contents

List of Figures	iv
Acknowledgment	v
1. Introduction	1
2. Approach/Concept	1
3. CORE Services	2
3.1 CORE Daemon.....	2
3.2 Hook Script.....	2
3.3 Service File.....	3
3.4 Draw_Links.py	5
4. Summary and Conclusion	6
5. References	7
Appendix A. One_Hop_Links CORE Service File	9
Appendix B. Draw_links Class File	11
Appendix C. IP-address-hook Script	17
Distribution List	19

List of Figures

Fig. 1 Runtime hook script	3
Fig. 2 Services on Node 3	4
Fig. 3 Edit service window	5

Acknowledgment

Jeff Ahrenholz, Boeing, provided much assistance in helping to understand the scope of the problem and provided direction in the early phases of this task. Mr Ahrenholz also provided some assistance with the IP-address-hook script that is included in this report.

INTENTIONALLY LEFT BLANK.

1. Introduction

The Network Science Research Laboratory (NSRL), formerly the Wireless Emulation Laboratory (WEL), of the US Army Research Laboratory (ARL) is able to conduct multiple experimentations that involve, but are not limited to, emulating a wireless tactical network environment. This is accomplished by running software applications to create the emulation environment under which experiments can be conducted. This includes some internal and external applications such as the Dynamically Allocated Virtual Cluster (DAVC), Extendable Mobile Ad-hoc Network Environment (EMANE), Common Open Research Emulator (CORE),¹ and the Scripted Display Tools (SDT3D).

One particular application that is being used is CORE (version 4.5svn4 [20130716]), a Python-based software application for building a representation of computer networks that runs in real-time. These networks can be visualized using CORE's graphical user interface (GUI) to draw nodes and the network connections that exist between them. There are two methods in which CORE calculates the network connectivity of nodes within the GUI: 1) using a distance-based formula or 2) using EMANE (version 0.8.1). By using EMANE, CORE takes advantage EMANE's ability to calculate node connectivity based on the waveform of the radio model used by each node along with that node's global positioning system (GPS) information.

The version of CORE that is being used only considers the distance-based formula to draw links on the GUI to represent network connectivity. This model works by calculating the pixel distance between two nodes positioned on the GUI surface then comparing the transposed pixel distance into meters whereby it can be compared to the maximum allowable distance between nodes. If the calculated distance is less than or equal to the maximum distance value, the link is drawn. Previously, there existed no method to subsequently draw those same links if EMANE is used to compute network connectivity.

This report describes a process for adding a service to the CORE application that will gather one-hop information and display that information on the CORE GUI and additionally on the SDT3D interface.

2. Approach/Concept

Because there is no existing method to extract link information from a CORE node when using EMANE network parameters, an approach to gathering node link information needed to be developed in order to draw the links using the CORE GUI and scripted display tool (SDT). The approach that was developed involved having each node obtain its route information using the

“ip” command, which is standard on most Linux distributions. Given the appropriate argument(s), this command can display the contents of the routing tables or the route(s) selected by some criteria.² Once the one-hop routes were determined, a structured command could be sent to the CORE GUI to draw the link between that node and each of the one-hop routes that exists. Given that the nodes are expected to be mobile, the routes could change over time, thus creating a need to periodically check the routing table for updates/changes. This would provide a method to check if previous routes were still present to determine if they needed to be removed from the CORE GUI.

3. CORE Services

In CORE, the concept of services is used to indicate what functionalities are enabled when a node is activated. These functionalities could be certain processes or scripts that are initialized or launched when the node is started. In order to incorporate a process to draw links when using EMANE as an available service to CORE nodes, the following elements within CORE needed to be modified or created: CORE daemon, Hook Script, Service File, and Draw_Links.py.

3.1 CORE Daemon

In order to create the One_Hop_Links drawing functionality as a service within CORE, there were a few modifications that needed to be made to the CORE code to allow usage of the existing methods that draw links under the basic range model. In particular, there were two files (`cored` and `nodes.py`) that needed to be updated in order to achieve this functionality. For `cored`, line 591 needed to be changed to the following line:

```
if not isinstance(netcommon, pycore.nodes.WlanNode) and \
    not isinstance(netcommon,
pycore.nodes.EmaneNode):
    continue
```

In the `nodes.py` code, line 65 needed to be changed to the following code:

```
def link(self, netif1, netif2):
    pass

def unlink(self, netif1, netif2):
    pass
```

3.2 Hook Script

Additionally, a hook script was created in order to provide the correct mapping of node name and IP address as they exist in the CORE environment when the emulation is started. Hook scripts are optional scripts that are run on each of the host when the emulation reaches a specified state. Currently, hook scripts can reside in the following six states: Definition,

Configuration, Instantiation, Runtime, Datacollect, and Shutdown. An example Runtime hook script is shown in Fig. 1.

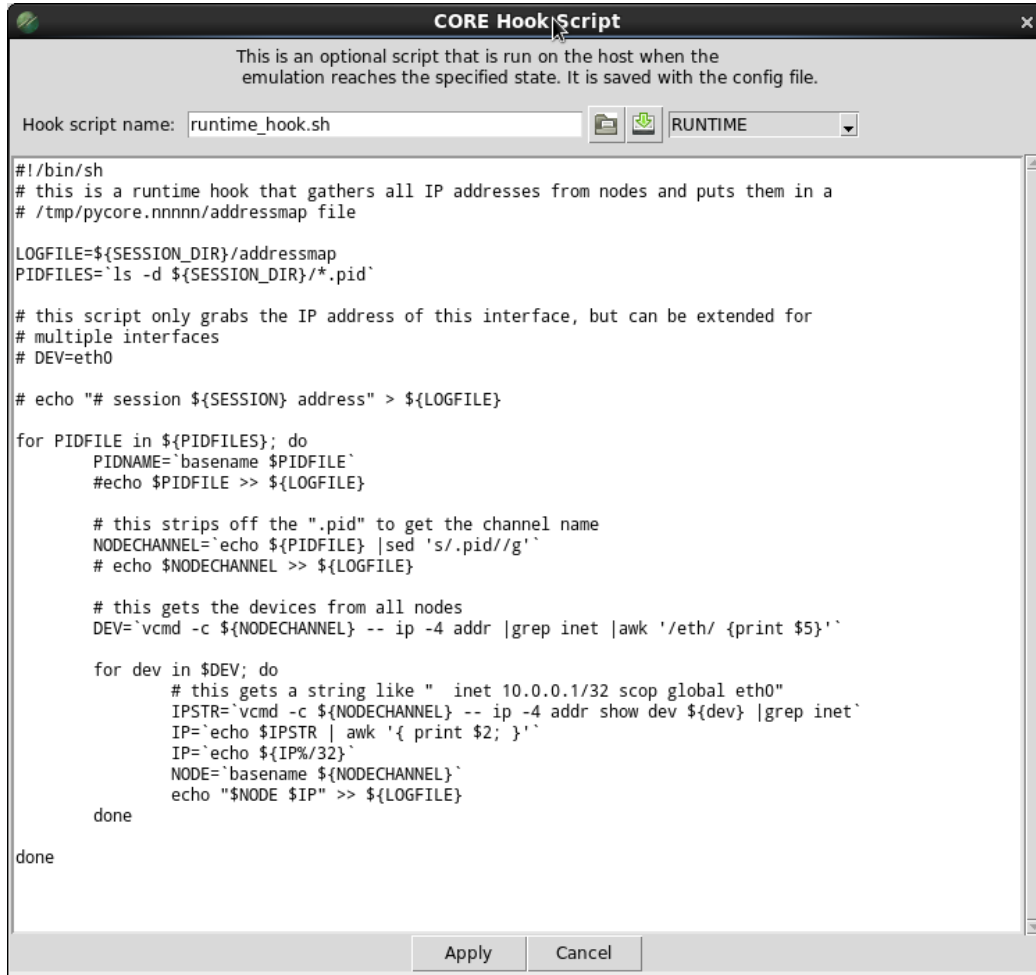


Fig. 1 Runtime hook script

The hook scripts shown in Fig. 1 gathers the IP address from each of the nodes in the CORE emulation environment along with its node ID and places it in a temp folder that gets creating when a CORE session is started. The temp folder is named according to the session ID; therefore, if the CORE session has an ID of “54356” the folder created for that session would be “/tmp/pycore.54356”. Within that folder, a file named “addressmap” would be created using the above hook script. The purpose behind creating this file in this location is so that it is accessible to all nodes during the time of the emulation. When the CORE session ends, this folder is removed along with all of its contents.

3.3 Service File

In order to get CORE to add a custom service as part of its set of available services, as pictured in Fig. 2, a service file must be created. Service files provide a method in which application can be launched on the nodes participating in the CORE emulation without having to log onto each

node individually. Custom service files can reside within any directory on the system, but the location of the custom files needs to be defined in the CORE configuration file (i.e., `/etc/core/core.conf`). This file is read each time the CORE daemon (`cored`) is started.

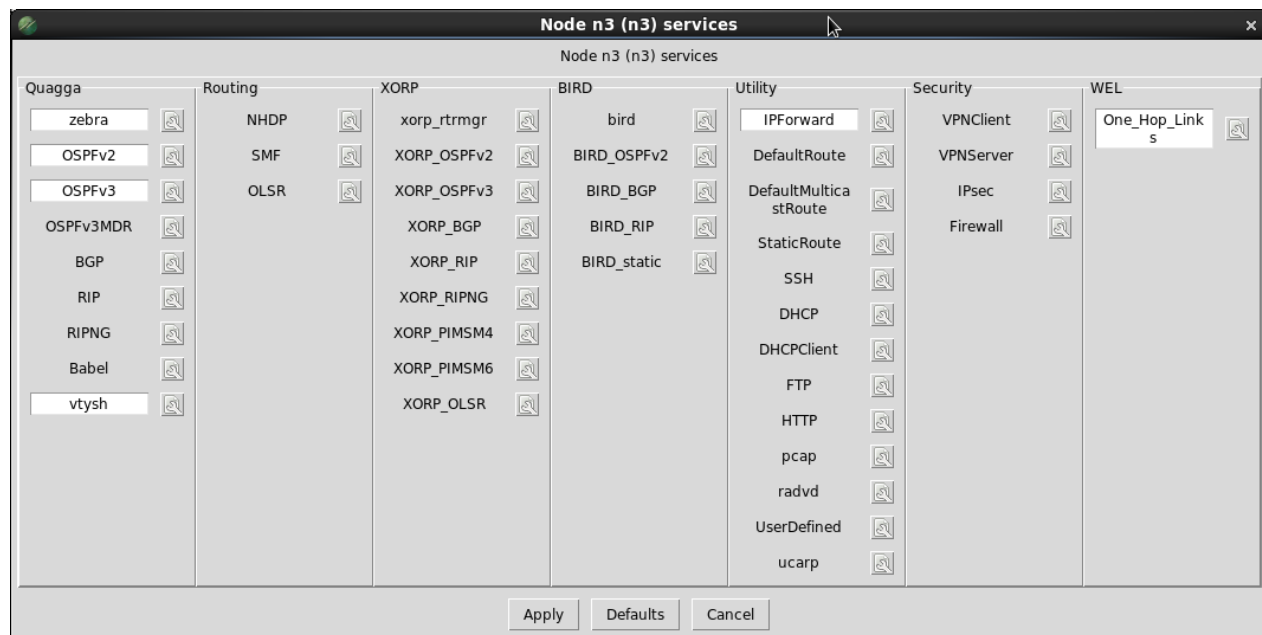



Fig. 2 Services on Node 3

Figure 2 shows which services or applications will be launched on Node 3 when the CORE emulation is started. The services that will be launched are highlighted when selected and the remaining services will not be activated even though they may be available for the node to use.

The service can be edited on each node individually by clicking on the wrench icon  that resides next to the service name. By clicking the wrench icon, a pop-up window appears showing what command will be executed on startup and shutdown of the emulation along with what additional files will be used and any per node directories the application needs. Figure 3 shows a depiction of the pop-up window. The operations that get populated into this window by the service is defined within the service file (Appendix A). This service file was created using examples provided by CORE and following the structure in which CORE calls its currently built-in services. The service `One_Hop_Links` starts a Python script called `draw_links.py`, which is explained in the next section.

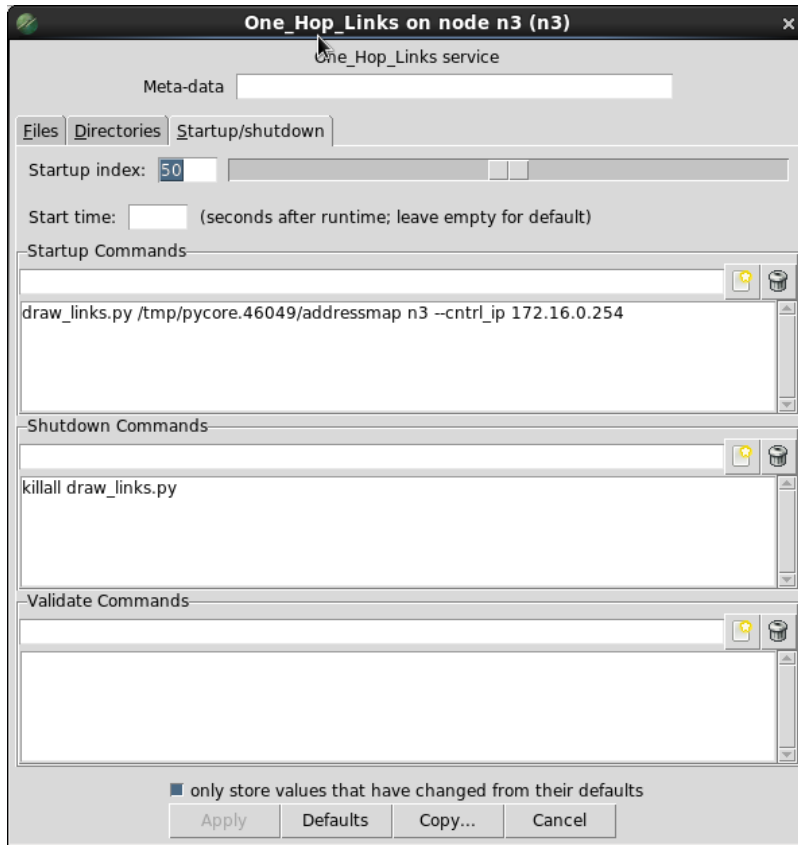


Fig. 3 Edit service window

3.4 Draw_Links.py

The Python script `draw_links.py` was created to read the route table of a node, determine the node's one-hop neighbors, and then send a message command to CORE to draw a line representing that link. The command is started on each node that has the `One_Hop_Link` service activated with the following three parameters: 1) the location of the addressmap file, 2) the id of the node, and 3) the IP of the CORE API host node. This script uses the addressmap file to correlate IP address listed in the route table with the correct node name in order to construct the proper message that needs to be sent to CORE to draw a link. The IP addresses that get selected from the routing table are determined based upon the value in the metric column. This value signifies the number of hops that it takes for the node to form a route to the selected IP. In our case, we specify two different metric values based upon the routing protocol that is used. If Open Shortest Path First (OSPF) is used, one-hop routes have a metric value of "2" while Optimized Link State Routing (OLSR) has a metric value of "1". The script currently determines which protocol is running on the node and using the metric value that corresponds to that protocol. There currently is not a default metric value if the node uses an alternate routing protocol besides the ones mentioned above. As other routing protocols get added to CORE, this feature can be adjusted to reflect those changes.

The message that the script construct to draw the link between two nodes consist of both end points of that link. The script uses the id of the node to determine the beginning end point while using the node name given by the addressmap file to determine the ending end point. The message that is send to CORE is shown below:

```
coresendmsg link flags=add type=0 n1number=2 n2number=3 -a
172.16.0.254
```

The sample message sends a command to the CORE application programming interface (API), residing at IP address 172.16.0.254, that adds a link between node 2 and node 3. In order to remove that link once the route goes away, the same message would be sent replacing `flags=add` with `flags=del`. By enabling the changes that were made to cored (CORE daemon) earlier in this report, when the message is sent to the CORE API the appropriate message is forwarded along to SDT as well.

4. Summary and Conclusion

The description of the CORE service that was defined in this report addresses a gap that existed within the CORE emulation environment where node links were not drawn when the EMANE network parameters were used. This report describes a method for determining routes between nodes based upon a metric value and displays those links on the CORE GUI and in SDT. This service can be started or stopped using the CORE services GUI and has been tested within the Network Science Research Lab.

5. References

1. Ahrenholz J. Comparison of CORE network emulation platforms. Proceedings of IEEE MILCOM Conference, 2010, pp.864–869.
2. Latvik M. ip(8) - Linux man page, n.d. [accessed 19Sept2013]. <http://linux.die.net/man/8/ip>.

INTENTIONALLY LEFT BLANK.

Appendix A. One_Hop_Links CORE Service File

The following is the One_Hop_Links CORE service file.

```
#####
# ARL
#
# author: Rommie Hardy <rommie.l.hardy.civ@mail.mil>
#####

'''
one_hop_links.py: pushes link message to cored to draw links on COR
GUI and SDT based on routing table.
'''

import os

class Draw_Links(CoreService):
    ''' Parent class for ARL services. Defines properties and methods
        common to ARL's processes.
    '''
    _name = "One_Hop_Links"
    _group = "WEL"
    _controlip = ("172.16.0.254", )
    _depends = ()
    _dirs = ()
    _configs = ()
    _startindex = 50
    _startup = ("draw_links.py", )
    _shutdown = ("killall draw_links.py",)

    @classmethod
    def getstartup(cls, node, services):
        ''' Return a string that will be written to filename, or sent
to the GUI for user customization.
        '''
        cmd = cls._startup[0]
        cmd += " %s/addressmap" % node.session.sessiondir
        cmd += " %s" % node.name
        cmd += " --cntrl_ip %s" % cls._controlip

        return (cmd, )

# this line is required to add the above class to the list of
available services

addservice(Draw_Links)
```

Appendix B. Draw_links Class File

The following is the Draw_links class file.

```
#!/usr/bin/env python

#####
# ARL
#
# author: Rommie Hardy <rommie.l.hardy.civ@mail.mil>
#
#####

import os, sys, time
import subprocess
import socket
import argparse

class Draw_links:
    def __init__(self ):
        '''
            Initialization
        '''
        self.metric = ''
        self.node_line = {}
        self.first = True

    def getAddressMap(self, AddressMap):
        '''
            Grab the address mapping key from
            /tmp/pycore.{Session}/addressmap
        '''
        self.node_list = {}
        map_file = open(AddressMap, "r")

        for line in map_file.readlines():
            node = line.split()[0]
            ip = line.split()[1]
            if not self.node_list.has_key(node):
                self.node_list[node] = [ip]
            else:
                self.node_list[node].append(ip)

        map_file.close()

        return self.node_list

    def getNodeID(self, node_ip):
        '''
            Determine the Node ID when an IP address is given
        '''
        for node, ip in self.node_list.iteritems():
```

```

        if node_ip.strip() in ip:
            return node
    return

def getNodeID_List(self, node_list):
    '''
    Determine the Node IDs when a list of IP addresses is given
    '''
    result_list = []

    for node in node_list:
        item = self.getNodeID(node)
        if(item):
            result_list.append(item)

    self.first = False

    return result_list

def checkServices(self):
    ''' Grab which services are running
        This is to determine which metric value to use
        OLSR: metric =1
        OSPF: metric =2
    '''
    DEVNULL = open(os.devnull, 'wb')

    cmd_ospf = "ps aux |grep ospf |grep -v grep"
    cmd_olsr = "ps aux |grep olsr |grep -v grep"

    ospf_status = subprocess.call(cmd_ospf, shell=True,
stdout=DEVNULL)
    olsr_status = subprocess.call(cmd_olsr, shell=True,
stdout=DEVNULL)

    if ospf_status == 0 and olsr_status == 1:
        self.metric = "2"
    elif ospf_status == 1 and olsr_status == 0:
        self.metric = "1"
    else:
        print "OLSR nor OSPF are running"
        exit()

    return self.metric

def getRoutes(self, metric):
    ''' Grab the info from the Routing table
    '''
    self.One_hop_routes = []
    cmd = 'ip -4 route |grep "metric %s" |awk \'{ print $1
}\'' % (metric)

```

```

        self.one_hop_list = os.popen(cmd)
        for node_ip in self.one_hop_list:
            self.One_hop_routes.append(node_ip)

        return self.One_hop_routes

    def addlink(self, node1, node2, cntrl_ip):
        ''' Create and send the message that CORE needs to add link
        between two nodes
        '''

        # attr = " line green"
        # self.session.sdt.cmd('link %s,%s%s' % (node1num, node2num,
        attr))
        # Directly send to SDT

        cmd = "coresendmsg link flags=add type=0 n1number=%s
n2number=%s -a %s" % (node1.lstrip('\n'), node2.lstrip('\n'), cntrl_ip)
        print cmd
        os.popen(cmd)

    def dellink(self, node1, node2, cntrl_ip):
        ''' Create and send the message that CORE needs to delete
        link between two nodes
        '''

        cmd = "coresendmsg link flags=del type=0 n1number=%s
n2number=%s -a %s" % (node1.lstrip('\n'), node2.lstrip('\n'), cntrl_ip)
        print cmd
        os.popen(cmd)

##### main body
#####

time.sleep(20)

Draw = Draw_links()

n=0

parser = argparse.ArgumentParser()
parser.add_argument("addressMap", help = "file created by CORE to map
node id to node ip in CORE")
parser.add_argument("node_id", help = "id of the node")
parser.add_argument("-c", "--cntrl_ip", help = "ip of the control
network for CORE on host", default = "172.16.0.254")

args = parser.parse_args()

while n != 5:

```

```

try:
    with open(args.addressMap) as file:
        break
except IOError as e:
    n +=1
    print "Looking for file, %s." %(args.addressMap)
    time.sleep(5)
finally:
    if n == 5:
        print "GIVING UP!!"
        print "File, %s, was not found!!" %(args.addressMap)
        exit()

first_run=True

# The variable "metric" could also be passed in from startup through
CORE services but requires a bit of investigation

metric = Draw.checkServices()

node_map = Draw.getAddressMap(args.addressMap)

if first_run:
    Old_OneHops = []

while True:
    try:
        OneHopList = Draw.getRoutes(metric)

        OneHop_NodeID_List = Draw.getNodeID_List(OneHopList)

        '''
        Delete a link
        '''
        for elem in set(Old_OneHops):
            if Old_OneHops.count(elem) !=
OneHop_NodeID_List.count(elem):
                print "Delete old link: ",elem
                Draw.dellink(args.node_id, elem, args.cntrl_ip)

        '''
        Add a new link
        '''
        for elem in set(OneHop_NodeID_List):
            if OneHop_NodeID_List.count(elem) !=
Old_OneHops.count(elem):
                print "Add new link: ",elem
                Draw.addlink(args.node_id, elem, args.cntrl_ip)

        Old_OneHops = OneHop_NodeID_List
        first_run = False
        time.sleep(1)

```

```
except KeyboardInterrupt:
    for elem in set(OneHop_NodeID_List):
        Draw.dellink(args.node_id, elem, args.cntrl_ip)
    exit()
```

Appendix C. IP-address-hook Script

The following is the IP-address-hook script.

```
#!/bin/sh
#####
#
# author: Jeff Ahrenholz, BOEING, <jeffrey.m.ahrenholz@boeing.com>
#
#
# modified by: Rommie Hardy <rommie.l.hardy.civ@mail.mil>
#
#####

# this is a runtime hook that gathers all IP addresses from nodes and
puts them in a
# /tmp/pycore.nnnnn/addressmap file

LOGFILE=${SESSION_DIR}/addressmap
PIDFILES=`ls -d ${SESSION_DIR}/*.pid`

# this script only grabs the IP address of this interface, but can be
extended for
# multiple interfaces
# DEV=eth0

# echo "# session ${SESSION} address" > ${LOGFILE}

for PIDFILE in ${PIDFILES}; do
    PIDNAME=`basename $PIDFILE`
    #echo $PIDFILE >> ${LOGFILE}

    # this strips off the ".pid" to get the channel name
    NODECHANNEL=`echo ${PIDFILE} | sed 's/.pid//g'`
    # echo $NODECHANNEL >> ${LOGFILE}

    # this gets the devices from all nodes
    DEV=`vcmd -c ${NODECHANNEL} -- ip -4 addr | grep inet | awk '/eth/
{print $5}'`

    for dev in $DEV; do
        # this gets a string like " inet 10.0.0.1/32 scop global
eth0"
        IPSTR=`vcmd -c ${NODECHANNEL} -- ip -4 addr show dev ${dev}
| grep inet`
        IP=`echo $IPSTR | awk '{ print $2; }'`
        IP=`echo ${IP%/32}`
        NODE=`basename ${NODECHANNEL}`
        echo "$NODE $IP" >> ${LOGFILE}
    done
done
```

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

2 DIRECTOR
(PDF) US ARMY RESEARCH LAB
RDRL CIO LL
IMAL HRA MAIL & RECORDS MGMT

1 GOVT PRINTG OFC
(PDF) A MALHOTRA

1 DIRECTOR
(PDF) US ARMY RESEARCH LAB
RDRL CIN T R HARDY

INTENTIONALLY LEFT BLANK.